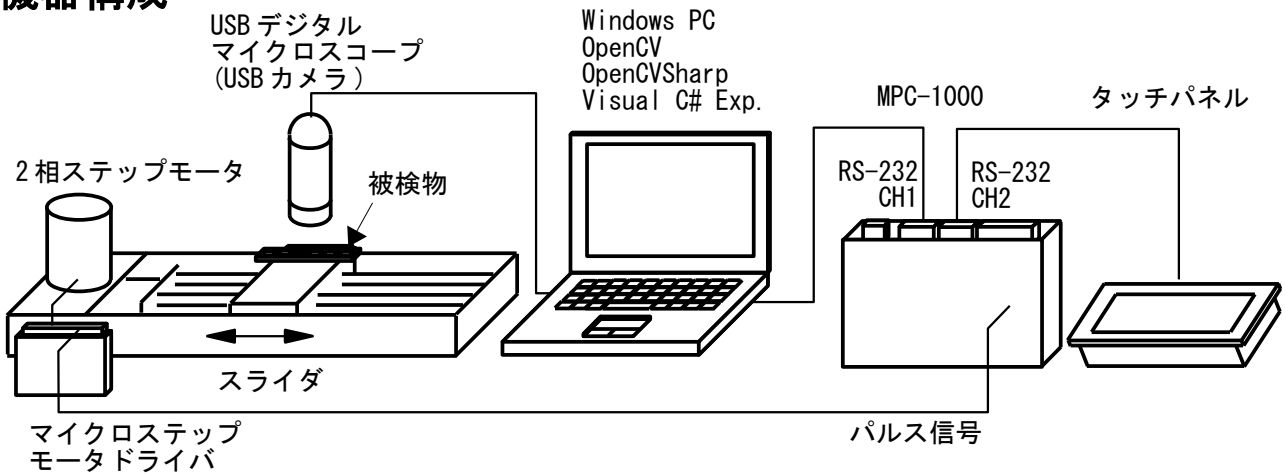
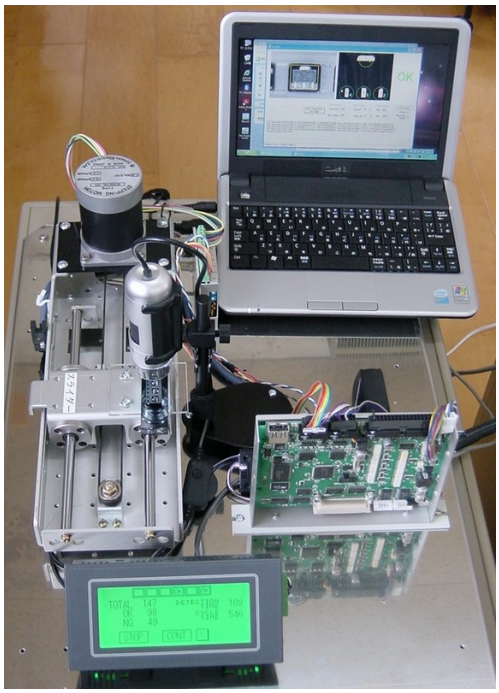
Application Note		Ref No: an2k-045	Last Modify 130802
テーマ	OpenCV を応用したデバイス方向検出実験 (Appendix: 色による領域検出)		
使用機器	MPC-1000,Windows PC,OpenCV,OpenCVSharp,Visual C# 2010 Express etc		

機器構成

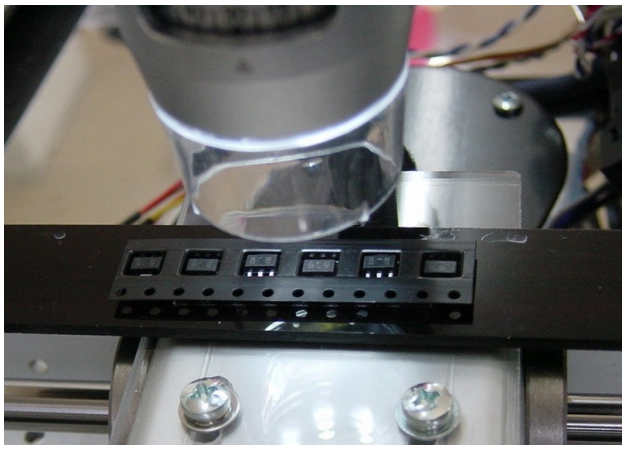


- トレイに入っている被検物を USB カメラでキャプチャ、画像処理をしてワークの天地を調べます。トレイはスライダで 1 コマずつピッチ送りをします。
- コントローラは MPC-1000 単体で、RS-232 通信、パルス発生をまかないます。RS-232 通信は MEWNET プロトコルを使用します。
- タッチパネルは TOTAL,OK,NG 個数、所要時間などを表示したり、ワーク位置合わせに用います。
- USB カメラはホビー用です。通販価格は約 8000 円でした。

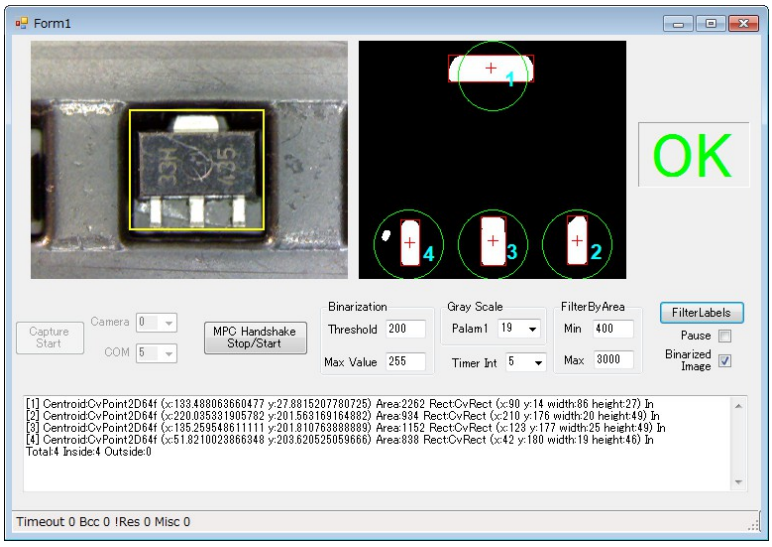


← 実験機

↓ ワーク
給材除材装置が無いので往復させてます。



検出例



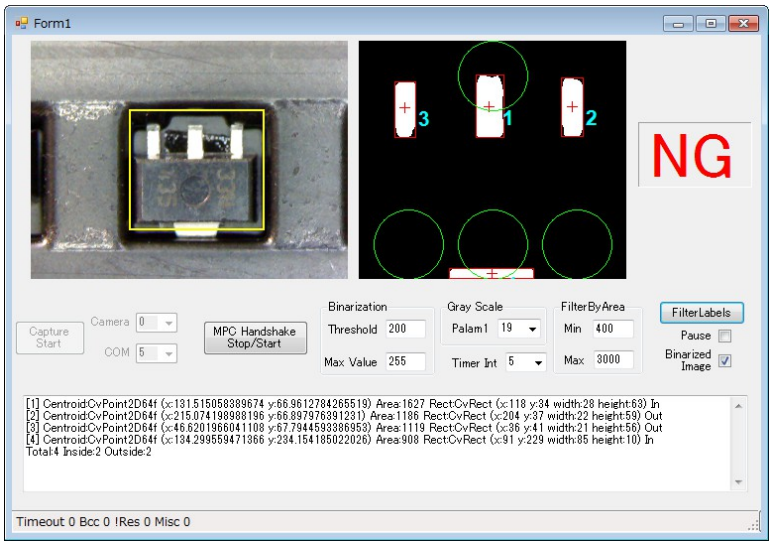
被検物（左画像）は非対称の IC です（4mm 角）。黄矩形内が検出範囲です。

OpenCV のライブラリを使って、グレースケール変換→スムージング→2 値化→ラベリング（FilterLabels）処理をすると、白色部分の面積、重心、矩形座標を得られます（右画像）。

所定範囲（緑丸）の中にある重心の数で IC の天地を判定をします。

白部分の面積はフィルタリングにより上限下限を設定しています。（4 番の左側の小さな白部分はレンジ外になります）

この場合は緑丸内の重心の数が 4 つなので OK です。



ワークが逆向きに入っていると緑丸内に収まる重心は少なくなります。

この場合は 2 つなので NG です。

グレースケール画面（右画像）。
検出されたエリアの位置関係がわかります。

MPC プログラム

```
  MEWNET 38400 1                /* PC と通信
  MEWNET 38400 2                /* Touch Panel Disply と通信
  MBK(8)=&H90                   /* タッチパネル画面切替
  OFF 71000                     /* PAUSE/CONT ボタン
  OFF 71002                     /* START/STOP ボタン
  PGA "A" 12000                 /* パルス出力加減速設定
  WAIT SW(204)==1
  PITCH=128                     /* work pallet pitch

  QUIT_FORK 1 *START_STOP

  END

*START_STOP                     /* Start/Stop、位置合わせ切替タスク
  DO
    IF SW(71002)==1 THEN        /* Touch Panel START/STOP alternate switch
      QUIT_FORK 2 *JOB
      WAIT SW(71002)==0
      QUIT 2
      OFF PGA
      TIME 100
    END_IF
    IF SW(73000)==1 THEN        /* Calibration
      GOSUB *CALIBRATION
      /* QUIT_FORK 2 *CALIBRATION
      /* WAIT SW(73000)==0     /* ページが変るまで結構回る
      /* TIME 500
    END_IF
    SWAP

  LOOP

*JOB

  OFF 70000                     /* 変換開始 off
  MBK(100)=0                    /* TOTALCount
  MBK(102)=0                    /* OK Count
  MBK(104)=0                    /* NG Count
  MBK(106)=0                    /* DETECTION TIME
  MBK(108)=0                    /* CYCLE TIME
  MBK(120)=0                    /* massage 0=Blank 1='Saved'
  OUT 0 72000                   /* OK/NG Lamp
  OK_COUNT=0                    /* OK Count of oneway
  NG_COUNT=0                    /* NG Count of one way

  DO

    GOSUB *HOME

    PGA "F" 10
    PGA "M" X(100)              /* Move to 1st Ponit. basically: X(100)=800
    WAIT SW(204)==1
    TIME 200                    /* カメラ安定待ち 原点から開始点まで移動するので最初は長い
                                /* PC の処理速度にも依存するようだ( Dell Mini WinXP は遅い)
    FOR j=0 TO 3

      OUT 0 72000
      OKLAMP_BIT=1
      FOR i=0 TO 5
        CYCLE_START=SYSCLK
        GOSUB *DETECT
        GOSUB *CYCLE_STOP
```

```
        PGA "R" PITCH
        WAIT SW(204)==1
        MBK(108)=SYSCLK-CYCLE_START  /* 1回の所要時間(移動時間含む)
        OKLAMP_BIT=OKLAMP_BIT<<1
      NEXT

      GOSUB *CYCLE_STOP
      TIME 200
      PGA "R" PITCH*-1
      WAIT SW(204)==1
      GOSUB *CHECK_OK_NG

      OUT 0 72000
      OKLAMP_BIT=&H20
      FOR i=0 TO 5
        CYCLE_START=SYSCLK
        GOSUB *DETECT
        GOSUB *CYCLE_STOP
        PGA "R" PITCH*-1
        WAIT SW(204)==1
        MBK(108)=SYSCLK-CYCLE_START
        OKLAMP_BIT=OKLAMP_BIT>>1
      NEXT

      GOSUB *CYCLE_STOP
      TIME 200
      PGA "R" PITCH
      WAIT SW(204)==1
      GOSUB *CHECK_OK_NG

    NEXT j

    PGA "F" 7

    FOR i=0 TO 2                 /* ダミーの動き(特に意味無し)
      PGA "M" 2200
      WAIT SW(204)==1
      /*TIME 50
      PGA "M" 100
      WAIT SW(204)==1
      /*TIME 50
    NEXT
    TIME 50

  LOOP

*DETECT                          /* リード検出
  TIME  100                      /* カメラ安定待ち
  DETECT_START=SYSCLK
  ON 70000                       /* start detection
  WAIT SW(70000)==0              /* wait detection end
  MBK(106)=SYSCLK-DETECT_START   /* 所要時間(通信時間も含む)
  SELECT_CASE MBK(1000)
    CASE 4: PRINT "OK"
            OK_COUNT=OK_COUNT+1
            MBK(102)=MBK(102)+1
            OUT OKLAMP_BIT|IN(72000) 72000
    CASE_ELSE : PRINT "NG"
            NG_COUNT=NG_COUNT+1
            MBK(104)=MBK(104)+1
  END_SELECT
  MBK(100)=MBK(100)+1

  RETURN

*HOME                            /* 原点復帰
```

```
  IF SW(192)==1 THEN
    PGA "F" 3
    PGA "R" 100
    WAIT SW(204)==1
    TIME 100
  END_IF

  PGA "G" -400
  WAIT SW(192)==1
  OFF PGA
  TIME 100      /* need
  PGA "H" 0
  RETURN

*CYCLE_STOP                     /* 一時停止
  IF SW(71000)==1 THEN
    WAIT (SW(71000)==0)|(SW(71001)==1)  /* 再開 or 1ピッチ送り
  END_IF
  RETURN

*CHECK_OK_NG                    /* 1スキャンのOK,NG数チェック(これは動作確認用)
  PR "OK" OK_COUNT "NG" NG_COUNT
  IF (OK_COUNT!=4)|(NG_COUNT!=2) THEN
    OFF 71002
    END
  END_IF

  OK_COUNT=0
  NG_COUNT=0
  RETURN

*CALIBRATION                    /* ワーク位置あわせ
  MBK(8)=&H91
  PGA "F" 10
  PGA "C"
  DO
    SELECT_CASE VOID
      CASE SW(73001)            /* HOME -> 1st
        GOSUB *HOME
        PGA "M" X(100)
        WAIT SW(204)==1
        PGA "C"
      CASE SW(73002)            /* >
        PGA "R" 1
        WAIT SW(204)==1
        PGA "C"
      CASE SW(73003)            /* <
        PGA "R" -1
        WAIT SW(204)==1
        PGA "C"
      CASE SW(73004)            /* SAVE
        X(100)=MBK(110)
        PR "SET X(100)=" V_PGA
        FSP
        MBK(120)=1
        TIME 1000
        MBK(120)=0
        WAIT SW(73004)==0
      CASE SW(73005)            /* RET
        MBK(8)=&H90
        RETURN
      CASE SW(73006)            /* >>
        PGA "R" PITCH
        WAIT SW(204)==1
        PGA "C"
      CASE SW(73007)            /* <<
```

```
        PGA "R" PITCH*-1
        WAIT SW(204)==1
        PGA "C"
      CASE_ELSE
    END_SELECT
    MBK(110)=V_PGA
    SWAP
  LOOP
```

VC# Express アプリケーション

- VC# Express で OpenCV を使うのに、Schima 氏が作成したラッパーライブラリ「OpenCVSharp」を用いました。
- OpenCV、OpenCVSharp などのインストールが必要です。
- Windows XP、7(x86)、7(x64) で動作を確認しました。
- アプリケーションのソースコードをご希望の方は技術までメールください。

タッチパネル画面

自動実行時画面

TOTAL,OK,NG 数，検出時間 (mSEC)，サイクルタイム (mSEC) 表示
開始・停止，一時停止・再開などの操作ができます。

位置あわせ画面

原点復帰，1 パルス送り，1 ピッチ送り，位置のフラッシュ ROM 保存ができます。

所要時間

- 検出時間 約 0.1 秒(MPC～PC 間通信も含む)。照明、PC の処理速度等に依存します。
- 1 サイクル 約 0.5 秒(移動含む)

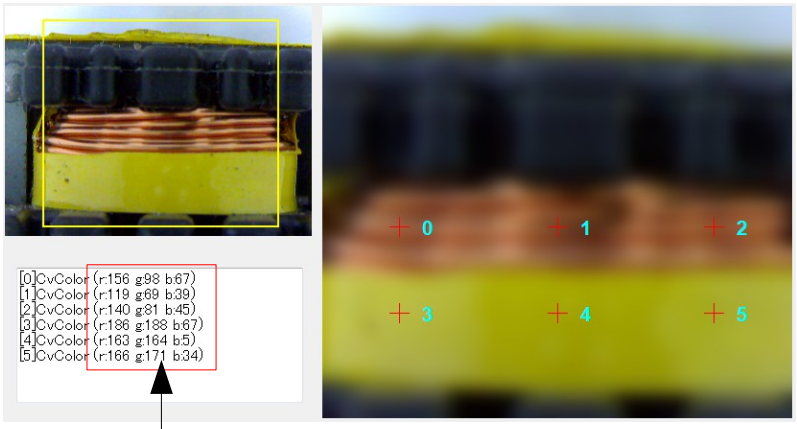
備考

- 使用した USB カメラはバカチョンなので条件設定ができません。移動後のカメラ安定待ちのため 100～200msec ディレイを入れてあります。工業用のカメラならこれが不要になるかも?
- 照明の明暗によってもキャプチャに要する時間が数 100msec 範囲で変ります。安定、高速化するには環境の整備が重要です。
- 環境、被検物の色、反射などで良好な画像を得られない場合は、OpenCV の他の機能(ピスセルアクセスやマッチング等)と併用するのも有効と思われます。このワークは本体とリードの反射率の差が大きくカメラ内蔵の LED 照明だけで結構安易に検出できました。

Appendix1

前述の内容とは関係ありませんが、色の識別を実験してみました。

元画像　　黄矩形部分を平滑化した画像



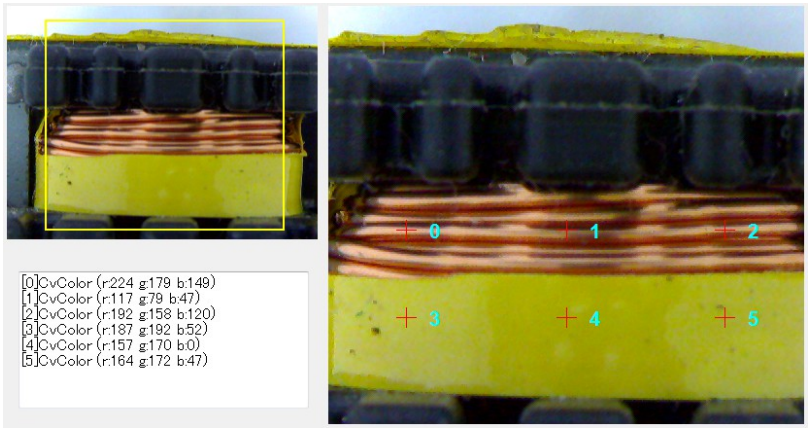
各ピクセルの RGB 値。

サンプルの被検物はコイル（エミフィル）です。コイルに巻かれている黄色のテープを半分剥がして、コイル部分とテープ部分が判別できるかを調べます。

カラー画像を平滑化して、特定ピクセル（画像 0 ～ 5 の十字交点）の RGB を求めます。

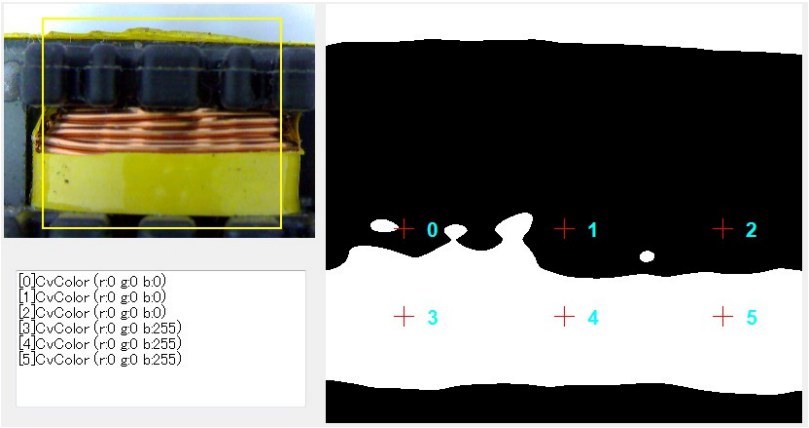
結果、G の値に差が出ました。
これを利用すればテープの有無をチェックすることが出来そうです。

【参考 1】　平滑化無しの場合



平滑化しないと、ピンポイントの反射や影の影響がもろに出ます。

【参考 2】　グレースケール変換 + 平滑化 +2 値化



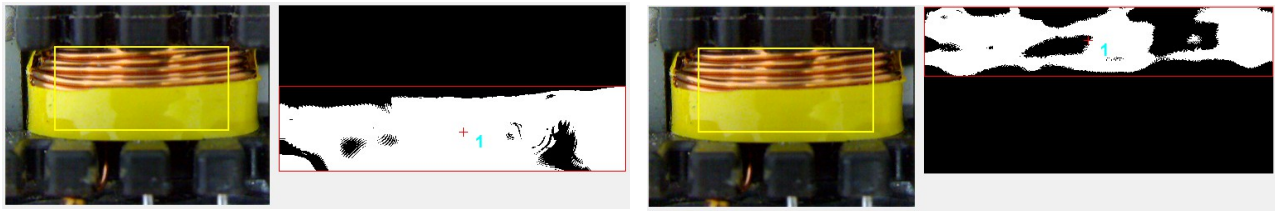
この被検物の場合は、2 値化でもテープの有無が検出できそうです。

Appendix2

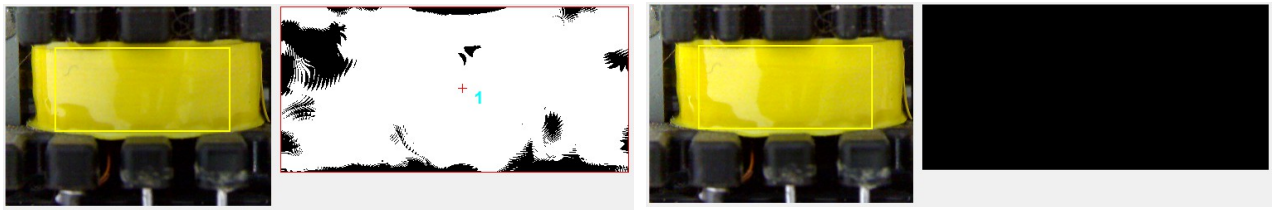
Appendix1 のテープ有無検出を色相による方法で試みました。
色相の範囲を限定し、テープ(または導線)部分を抽出→2 値化→ラベリングを行います。
検出数、面積、矩形サイズでテープの有無を確認できそうです。

左列はテープ色に合わせた色相(28～32 度)、 右列はコイルに合わせた色相(10～15 度)で処理をしました。 (OpenCV は通常の色相の 1/2 の値)

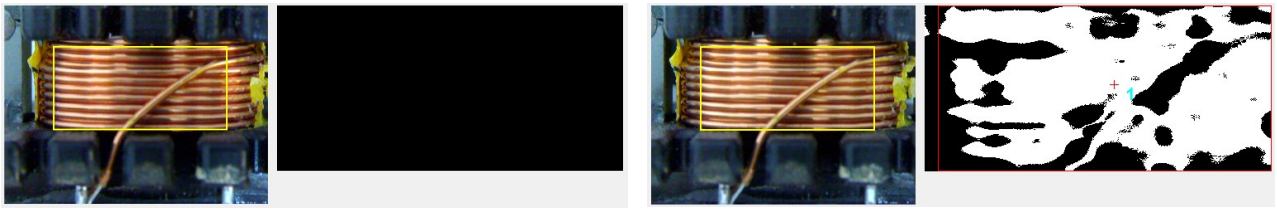
半分だけテープ有り。



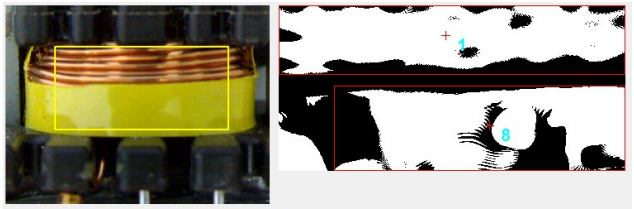
全面テープ有り。



テープ無し。



色相の範囲を広げると（10 ～ 32 度）両方拾う。



--- End Of Doc ---